

Documentation for random.h and random.c

Steven Andrews, © 2003

See the document “LibDoc” for general information about this and other libraries.

```
#ifndef __random_h
#define __random_h

#include <time.h>
#include <stdlib.h>
#include <math.h>

#define RAND_MAX_30 1073741823
#if RAND_MAX==32767
    #define rand30() ((long)rand()<<15|rand())
#elif RAND_MAX<RAND_MAX_30
    #define rand30() ((rand()&32767L)<<15|rand()&32767L)
#else
    #define rand30() (rand()&RAND_MAX_30)
#endif

#define exprand(a) (-log(((float)rand()+1.0)/(RAND_MAX+1.0))*(a))
#define exprand30(a) (-log(((double)rand30()+1.0)/(RAND_MAX_30+1.0))*(a))
#define unirand(lo,hi) ((float)rand()/RAND_MAX*((hi)-(lo))+(lo))
#define unirand30(lo,hi) ((double)rand30()/RAND_MAX_30*((hi)-(lo))+(lo))
#define signrand() (rand()&1?1:-1)
#define coinrand(p) (rand()<(RAND_MAX+1.0)*(p))
#define coinrand30(p) (rand30()<(RAND_MAX_30+1.0)*(p))
#define intrand(n) (rand()%(n))
#define intrand30(n) (rand30()%(n))
#define thetarand() (acos(1.0-2.0*rand()/RAND_MAX))
#define radrand1(r) ((r)*sqrt((float)rand()/RAND_MAX))
#define radrand2(r) ((r)*pow((float)rand()/RAND_MAX,0.333333333333))
#define powrand(xmin,power)
    (xmin*pow(((float)rand()+1.0)/(RAND_MAX+1.0),1.0/(1.0+power)))
#define powrand30(xmin,power)
    (xmin*pow(((double)rand30()+1.0)/(RAND_MAX_30+1.0),1.0/(1.0+power)))

unsigned int randomize();
float binomrand(int n,float m,float s);
int intrandp(int n,float *p);
int poisrand(float xm);
float binomialrand(float p,int n);
float gaussrand();
void sphererand(float *x,float rad1,float rad2);
void sphererandd(double *x,double rad1,double rad2);
void randtable(float *a,int n,int eq);
void randshuffletable(float *a,int n);
void showdist(int n,float low,float high,int bin);

#endif
```

Include: <stdio.h>, <time.h>, <math.h>
 Example program: LibTest.c

History: Written 5/12/95; modified 11/12/98. Routines have been moderately tested. Works with Metrowerks C. Documentation updated 10/16/01. Ported to Linux 10/16/01. Added randtable 11/16/01. Added intrandp 1/14/02. Added poisrand 1/26/02. Added rand30 and other ...30 functions 11/8/02, but didn't document them until 9/2/03. Added gaussrand 4/24/03. Added intrand30, coinrand30, RAND_MAX_30, and improved unirand and unirand30 9/2/03. Added radrand1 3/11/04. Added radrand2 and sphererand 3/25/04. Replaced a few implicit type casts with explicit ones 6/9/04. Added randshuffletable 7/21/05. Changed randomize() from a macro to a function 8/26/05. Added binomialrand 3/28/06. Added powrand and powrand30 11/13/06.

Most of these routines return random numbers, chosen from a variety of densities. They use the stdlib.h rand() function, and have not been analyzed for the randomness quality. Before using these routines, it is recommended that the random number generator seed be set with either the stdlib.h function srand(unsigned int seed) or set to the clock value with randomize.

Note that $1.0 * \text{rand}() / \text{RAND_MAX}$ returns a uniform density on $[0,1]$ and $(\text{rand}() + 1.0) / (\text{RAND_MAX} + 1.0)$ is uniform on $(0,1]$. To convert these uniform densities to the density $f(x)$, first calculate the cumulative probability $P(x) = \int_{-\infty}^x f(x') dx'$, where it is seen that $P(x)$ is 0 at $x = -\infty$ and 1 at $x = \infty$. Now if the value for $y = P(x)$ is chosen with a uniform density, its value mapped onto x has the desired density. Thus a function should return $x = P^{-1}(y)$. It is also helpful to know that the CodeWarrior compiler on a Macintosh has RAND_MAX equal to $2^{15} - 1$, whereas it is $2^{31} - 1$ for the gcc compiler on Linux. The routines that end with a "30" are especially helpful on a Macintosh (and hinder slightly on Linux) by allowing 2^{30} possible random numbers. For the most part, I have not had any trouble with the randomness quality on a Macintosh, although in one instance I found net diffusion of randomly moving particles towards the center of the volume; this undoubtedly arose from an imperfect random number generator, although the precise problem is unclear. I solved it by shuffling the lookup table that I was using.

The table below shows the domain, range, and densities of the macros and routines given here. The domains are the domains over which the functions give reasonable values, but are not necessarily sensible. For example, coinrand can accept an input anywhere between $-\infty$ and ∞ , although the function always returns 0 if $p < 0$ and 1 if $p > 1$. The densities are only strictly correct in the limit that RAND_MAX approaches infinity. In regions where the density is small (where $f(x)\Delta x \approx 1/\text{RAND_MAX}$, for some characteristic Δx), a small set of random numbers is mapped to a large output range, leading to relatively sparse coverage.

Name	Domain	Range	Density
exprand	$[0, \infty)$	$[0, \infty)$	$1/a * \exp(-x/a)$
	$(-\infty, 0]$	$(-\infty, 0]$	$1/a * \exp(-x/a)$
exprand30	$[0, \infty)$	$[0, \infty)$	$1/a * \exp(-x/a)$
	$(-\infty, 0]$	$(-\infty, 0]$	$1/a * \exp(-x/a)$
unirand	$(-\infty, \infty); lo \neq hi$	$[lo, hi]$	$1/hi - lo$
unirand30	$(-\infty, \infty); lo \neq hi$	$[lo, hi]$	$1/hi - lo$
signrand		$\{-1, 1\}$	$\{0.5, 0.5\}$
coinrand	$(-\infty, \infty)$	$\{0, 1\}$	$\{1-p, p\}$
coinrand30	$(-\infty, \infty)$	$\{0, 1\}$	$\{1-p, p\}$
intrand	$[1, \infty)$	$\{0, 1, \dots, n-1\}$	$\{1/n, 1/n, \dots, 1/n\}$
intrand30	$[1, \infty)$	$\{0, 1, \dots, n-1\}$	$\{1/n, 1/n, \dots, 1/n\}$

thetarand		$[0, \pi]$	$1/2 * \sin(x)$
radrand1	$(-\infty, \infty)$	$[0, r]$	$2x/r^2$
radrand2	$(-\infty, \infty)$	$[0, r]$	$3x^2/r^3$
powrand	$(-\infty, \infty), (-\infty, -1)$	$[x_{min}, \infty)$	$(1-m)/x_{min} * (x/x_{min})^m$
powrand30	$(-\infty, \infty), (-\infty, -1)$	$[x_{min}, \infty)$	$(1-m)/x_{min} * (x/x_{min})^m$
binomrand	$n > 0$, all m, s	$[m-s\sqrt{(3n)}, m+s\sqrt{(3n)}]$	\approx Gaussian with mean m , std. dev. s
intrandp	$n > 0$, $0 \leq p_i \leq 1$	$\{0, 1, \dots, n-1\}$	$\{p_0, p_1 - p_0, \dots, 1 - p_{n-2}\}$
poisrand	$(-\infty, \infty)$	$[0, \infty)$	Poisson with mean xm
binomialrand	$[0, 1], [0, \infty)$	$[0, n]$	Binomial deviate, prob. p , n trials
gaussrand		$(-\infty, \infty)$	Gaussian with mean 0, std. dev. 1
sphererand	$[0, \infty)^2$	$[-rad2, rad2]^3$	Point in spherical shell
sphererandd	$[0, \infty)^2$	$[-rad2, rad2]^3$	Point in spherical shell

randomize

This sets the random number generator seed to the current time and also returns the seed that was used.

exprand

This returns an exponentially distributed random double.

exprand30

This is identical to exprand, except it uses a 30 bit random number.

unirand

This returns a uniformly distributed double between l_o and h_i , inclusive.

unirand30

This is identical to unirand, except it uses a 30 bit random number.

signrand

This returns 1 or -1 with equal probability.

coinrand

This returns 1 with probability p , and 0 otherwise.

coinrand30

This is identical to coinrand, except it uses a 30 bit random number.

intrand

This returns an integer between 0 and $n-1$ with equal probability for each value. The probability distribution is correct if n is a divisor of $RAND_MAX+1$ (*i.e.* an integer power of 2), quite good if n is a small integer, and poor if n is a significant fraction of $RAND_MAX$ and not a divisor of $RAND_MAX+1$.

intrandp

This is identical to intrand, except it uses a 30 bit random number.

thetarand

This is intended for use in choosing a random $\hat{\mathbf{r}}$ direction in spherical coordinates.

radrand1

This is intended for use in choosing a random radius within a circle of radius r . In combination with a random angle (uniform between 0 and 2π), this yields a random point uniformly distributed within the circle.

radrand2

This is intended for use in choosing a random radius within a sphere of radius r . In combination with a random spherical angle, this yields a random point uniformly distributed within the sphere.

powrand

This returns a random number chosen from a power law distribution with slope m , which needs to be <-1 . x_{\min} is typically positive, in which case it is the smallest number that can be returned; it can also be negative, which just switches the sign of the returned value.

powrand30

This is identical to `powrand`, but for 30-bit random numbers. It is advised for accurate work because of the very long tail of the distribution.

binomrand

This adds together n random variables from a uniform density and then scales the sum to yield the proper mean and standard deviation. It's an easy alternative for a true Gaussian density, although not as fast or as well distributed as a look-up table and interpolation (see `randtable`). It's also misnamed, since a true binomial distribution is the sum of numbers chosen from $\{0,1\}$.

intrandp

This is similar to `intrand`, but allows non-uniform probabilities for each integer (however, it doesn't improve on the distribution accuracy for large n values). p is sent in as a list of cumulative probabilities for each integer. Since they are cumulative, p is an increasing list of numbers between 0 and 1, and p_{n-1} should be equal to 1. Results will always be between 0 and $n-1$, even with incorrect p values.

poisrand

This returns an integer chosen from a Poisson density with mean x_m , which will typically be in the range $x_m \pm \sqrt{x_m}$. This routine is copied almost verbatim from *Numerical Recipes*. A feature which the book routine has and which is kept here is that if the routine is called more than once with the same value of x_m , it doesn't recalculate some variables, in order to speed up the routine. Negative values of x_m are possible but always return a value of 0.

binomialrand

This returns a random integer (as a float) chosen from a binomial distribution for n trials, each with probability p . It is the number of successes for these n trials. The routine was copied nearly verbatim from *Numerical Recipes*.

gaussrand

This returns a normal deviate with mean 0 and standard deviation 1 using the Box-Muller transformation described in *Numerical Recipes*.

sphererand

This returns a 3 dimensional point in x which is uniformly distributed within a spherical shell bounded on the inside by `rad1` and the outside by `rad2` (both inclusive). For a fixed radius, set both `rad1` and `rad2` to the radius. The input contents of x are ignored although it needs to be allocated to at least size 3.

sphererandd

This is identical to `sphererand` except that it uses doubles rather than floats.

`randtable`

This fills in a lookup table with entries for quickly converting a uniform density to an alternate density, using `eq` to indicate which density is desired. `n` is the number of elements in the table. If `eq` is 1, the density is a normal density with mean 0 and standard deviation 1; returned values range from $-\text{erf}^{-1}(0.5/n-1)$ to $\text{erf}^{-1}(0.5/n-1)$. For example, if `rt` is a table with 1024 elements, the following expression would return a normally distributed random variable with mean `mu` and standard deviation `sd`: `x=mu+sd*rt[rand()&1023]`, also the range is from -2.33 to 2.33 . Clearly, there are only `n` possible outcomes in this expression, which could be corrected by linear interpolation and somewhat slower and lengthier code.

`randshuffletable`

This shuffles a list of `n` numbers such that each number is equally likely to end up at any position in the list.

`showdist`

This is only intended for debugging routines such as `binomrand`, so it is not a general routine. It plots a bar graph (bin bars that range from the first bar center at low to the last bar center at high) showing the distribution of `n` random variables from `binomrand(10,0,1)` or some other routine; it also displays the actual mean and standard deviation.